**Echo – Social Media Application   
Technical Design Document**

**Technical Design Document: Echo Social Media Application**

**1. Introduction**

**1.1 Purpose**

This Technical Design Document (TDD) outlines the technical architecture, design patterns, data models, and technology stack required to implement the Echo Social Media Application as described in the Functional Requirements Document (FRD). It serves as a blueprint for the development team to build, test, and deploy a scalable, secure, and engaging social media platform.

**1.2 Intended Audience**

This document is intended for:

* Frontend and Backend Developers
* Database Administrators
* DevOps Engineers
* Software Architects
* Technical Leads
* Quality Assurance Engineers
* All stakeholders involved in the design, development, and deployment of the Echo platform will benefit from this document.

**1.3 Scope**

This TDD covers both high-level and low-level technical design aspects of the Echo platform, including:

* User Management
* Content Creation and Feed Management
* Exploring other users
* Relationship between the users(Following and Followers)
* Posting stories.

It also addresses non-functional requirements such as performance, scalability, security, usability, and maintainability.

**1.4 Relationship to FRD**

This document directly references and expands upon the functional and non-functional requirements defined in the FRD. Each technical design decision is made to fulfill one or more of the stated requirements, ensuring that the final implementation aligns with the product vision and user expectations.

**2. High-Level Architecture**

We propose a multi-tier architecture to ensure scalability, maintainability, and security. The key layers are:

**2.1 Presentation Layer**

This layer will be responsible for the user interface (UI) and user experience (UX). It will handle user interactions and display information to various user roles.

* **Technology:** React, HTML5, CSS3,SCSS and Material UI.
* **Key Components:**
  + User registration and login interfaces.
  + Updating user profile.
  + Post creation and interaction interfaces (likes, comments).
  + Search and profile browsing components.
  + Relationship between the users(Following and Followers).
  + Posting stories.

**2.2 Application Layer (API Layer)**

This layer will act as an intermediary between the presentation layer and the business logic layer. It will expose RESTful APIs for the UI to interact with the backend.

* **Technology:** Java (Spring Boot), React (Axios)
* **Key Components:**
  + API endpoints for user Log In and user Sign Up.
  + API endpoints for home page content.
  + API endpoints for user profile.
  + API endpoints for creating a post.
  + API endpoints for searching a profile.

**2.3 Business Logic Layer**

This layer will contain the core business rules and logic of the social media platform. It will handle data validation, content management, user interactions, and interactions with the data access layer.

* **Technology:** Same as the Application Layer (will reside within the same application or as separate microservices).
* **Key Components:**
  + User Management
  + Content Creation and Feed Management
  + Relationship between the users(Following and Followers
  + Explore and Search Features
  + Posting stories

**2.4 Data Access Layer**

This layer will be responsible for interacting with the database and managing data persistence.

* **Technology:** MySQL
* **Key Components:**
  + Data models for users, posts, likes, comments, relationship(followers and following), story.

**3. Data Model**

The data model will be designed to efficiently store and retrieve information related to users, posts, comments, messages, notifications, and hashtags. Below is a high-level overview of key entities and their attributes:

* **User:**
  + id : BIGINT, PRIMARY KEY, AUTO\_INCREMENT
  + name: VARCHAR(255)
  + username: VARCHAR(255), UNIQUE, NOT NULL
  + email: VARCHAR(255), UNIQUE, NOT NULL
  + password: VARCHAR(255), NOT NULL
  + profile\_pic: VARCHAR(255), DEFAULT NULL
  + cover\_pic: VARCHAR(255) DEFAULT NULL
  + city: VARCHAR(255), DEFAULT NULL
  + websiteName: VARCHAR(255), DEFAULT NULL
  + websiteURL: VARCHAR(255), DEFAULT NULL
* **Posts:**
  + id: BIGINT, PRIMARY KEY, AUTO\_INCREMENT
  + user\_id (Foreign Key):BIGINT, NOT NULL, FOREIGN KEY referencing Users.user\_id
  + Description:TEXT,DEFAULT NULL
  + Image:VARCHAR(255),DEFAULT NULL
  + Created\_at:DATETIME(6),NOT NULL
* **Comments:**
  + id : BIGINT, PRIMARY KEY, AUTO\_INCREMENT
  + Created\_at:DATETIME(6),NOTNULL
  + Description:TEXT,DEFAULT NULL
  + post\_id: BIGINT, NOT NULL
  + user\_id (Foreign Key): BIGINT, NOT NULL
* **Likes:**
  + Post\_id:BIGINT,PRIMARY KEY
  + User\_id:BIGINT,PRIMARY KEY
* **Relationship:**
  + Followed\_user\_id:BIGINT,PRIMARY KEY
  + Follower\_user\_id:BIGINT,PRIMARY KEY
* **Stories**
  + Id:BIGINT,PRIMARY KEY,AUTO\_INCREMENT
  + Created\_at:DATETIME(6),NOT NULL
  + Image:VARCHAR(255),DEFAULT NULL
  + User\_id:BIGINT,NOT NULL

**4. API Design**

We will adopt a RESTful API design for communication between the frontend and backend. API endpoints will be logically grouped by resource. Examples include:

* **User APIs:**
  + POST /api/auth/register: Register a new user (name, email, password, etc.).
  + POST /api/auth/login: Authenticate user and return a JWT token.
  + POST /api/auth/logout: Removes the token from the cookies and logs out the user
  + GET /api/users/find/{userid}: Retrieve a user's profile by their ID.
  + PUT /api/users: Update a user's profile (name, bio, profile picture, password, etc.).
  + DELETE /api/users/{userid}: Delete a user's account (protected).
  + GET /api/users/suggestions: Get other users in the suggestion section from the database
  + GET /api/users/search: Find user by username or name
  + GET /api/users/check-username: Check whether the username is already taken or not
  + GET /api/users/check-email: Check whether a user exists with the same email or not
* **Post APIs:**
  + POST /api/posts: Create a new post (requires authentication).
  + POST /api/posts/upload: Upload a photo in the post if it contains one.
  + GET /api/posts: Retrieve all posts from the current user.
  + DELETE /api/posts/{id}: Delete a post (requires the author's authentication).
* **Comment APIs:**
  + POST /api/comments: Add a new comment to a specific post.
  + GET /api/posts: Retrieve all comments for a specific post when post id sent as query parameter.
  + DELETE /api/comments/{ id}: Delete a comment (requires the comment author).
* **Like APIs:**
  + - POST /api/likes: Add a like to a post.
    - GET /api/likes :Get the list of likes for a post
    - DELETE /api/likes :Delete the like for a post
* **Relationships APIs:**
  + POST /api/relationships : For following a user
  + GET /api/relationships/following/count : For getting the number of users the current user is following
  + GET /api/relationships/followers/count: For getting the number of users who follows the current user.
  + GET /api/relationships/following/list: Get the list of user who the current user is following
  + GET /api/relationships/followers/list: Get the list of users who are following the current user.
  + GET /api/relationships : Check whether the current user is following another user or not
  + DELETE /api/relationships : Deletes the relationship between the two users i.e. unfollow function
* **Stories:**
  + POST /api/stories: To add a story
  + GET /api/stories: To get the stories of the other user that the current user would be able to view.
  + DELETE /api/stories/{id}: To delete the story posted by the current user.

API requests and responses will use JSON format. We will implement appropriate HTTP methods (GET, POST, PUT, DELETE) based on the action being performed. Authentication will be handled using JWT (JSON Web Tokens), and authorization will be role-based.

**5. Technology Stack**

The following is a preliminary technology stack proposal:

* **Frontend:** React, HTML5, CSS3,SCSS and Material UI (as specified in the FRD for UI components)
* **Backend:** Java (Spring Boot)
* **Database:** MySQL

**6. Detailed Design – Key Features**

**6.1 User Management**

* **Registration:** Implement separate registration flows for general users.
* **Login:** Secure login using email/username and password, with an option for Google OAuth integration as per FRD.
* **Profile Management:** Users can view and update their profile details (name, profile picture, cover picture, location and any website links) and manage privacy settings.

**6.2 Content Management**

* Users can create, edit, and delete posts with text and images.
* Posts support engagement features: likes, comments and likes.
* Users can establish relationships with other users with the help of follower-following feature.
* Users can view their own as well as their following’s story.

**7. Deployment Architecture**

**7.1 Application Servers**

* Host the backend logic for user authentication, post management, likes, comment and relationship management.
* Built using scalable backend frameworks (e.g., Spring Boot).
* Deployed in auto-scaling groups to dynamically adjust the number of instances based on user activity and traffic spikes.

**7.2 Frontend Hosting**

* The frontend (built with React) is hosted on an instance of EC2 for fast, global access.
* Static assets (HTML, CSS, JS, images) are cached and served from edge locations to reduce latency and improve load times.

**7.3 Database Servers**

* Use a managed relational database service (e.g., MySQL) to store user profiles and settings, posts, comments, likes, and relationship management(Followers and Followings).
* Implement read replicas to scale read-heavy operations like feed rendering and search queries.

**7.4 Media Storage**

* Store user-uploaded content (images, videos) in object storage (e.g., Amazon S3 or Firebase Storage).
* Use EC2 integration for fast and secure media delivery.
* Generate signed URLs for secure access to private media files.

**7.5 Security & Monitoring**

* All communications are secured via HTTPS.
* Use JWT or OAuth 2.0 for secure session management.
* Implement robust logging and monitoring solutions to track application performance, errors, and security events.

**8. Conclusion**

This Technical Design Document provides a comprehensive overview of the technical architecture and design considerations for the Echo Social Media Application. It outlines the proposed technology stack, data model, API design, and deployment strategy, all while addressing the functional and non-functional requirements outlined in the FRD. This document will serve as a guiding framework for the development team to build a robust, scalable, secure, and user-friendly social media platform. This is a living document and will be updated as the project progresses and more detailed design decisions are made.

**Workflow:** REGISTER -> LOGIN -> HOME -> CREATE PROFILE -> VIEW PROFILE -> EDIT PROFILE -> CREATE POST -> VIEW POSTS -> COMMENT ON POSTS -> LIKE ON POSTS -> LOGOUT